Slide 1

Administrivia

- Homework 1 graded; sample solution on the Web.
- Homework 3 will be on the Web tomorrow, due next Monday.

Slide 2

Strings in C — Review/Recap

- Many languages have nice ways of working with text (character strings). What C provides is — no surprise — somewhat primitive.
- In C, strings are arrays of `char`s, with the convention that the actual text of interest is followed by a null character (8-bit zero, represented in code as `'\0'`.

Slide 3

Sidebar — Variables and Memory

- Many languages provide a rather abstract view of data/variables, which is convenient but hides the low-level workings.
- C, in contrast, typically exposes a much lower-level view: Memory is a big one-dimensional space, typically partitioned into small units (bytes) consisting of binary digits (bits). When you declare a variable, the compiler reserves some space (amount determined by the variable's type) and associates it with the variable's name and type. The name provides a way to access that piece of memory; the type determines how the bits will be interpreted (as a binary integer, as an array of ASCII characters, etc.).

  (I say "typically" here because there are probably exceptions — there are C compilers for a *lot* of systems, some of them fairly offbeat by current standards.)

Slide 4

Pointers in C — Review/Recap

- C, in contrast to Python and Scala, makes an explicit distinction between things and pointers-to-things. Pointers-to-things are essentially memory addresses, though usually declared to point to variables (or data) of a particular type. (Exception: a "void" pointer can point to data of any type. See `man` page for `memcpy` for example.)
- Two useful operators are `&` ("address of") and `*` (dereference).
- In C, pointers and arrays are in some sense(s) equivalent — not identical, but in many contexts interchangeable. This is reflected in the `man` pages for many functions (e.g., `printf`).

Slide 5

Parameter Passing in C

- In C, all function parameters are passed "by value" — which means that the value provided by the caller is copied to a local storage area in the called function. The called function can change its copy, but changes aren't passed back to the caller.
- An apparent exception is arrays — no copying is done, and if you pass an array to a function the function can change its contents (as we did in the sort program). Why "apparent exception"? because really what's being passed to the function is not the array but a pointer! so the copying produces a second pointer to the same actual data.
- This is at least simple and consistent, but has annoying limitations . . .

Slide 6

Pass By Reference (Sort Of)

- A significant potential limitation on functions is that a function can only return a single value. Pointers provide a way to get around this restriction: By passing a pointer to something, rather than the thing itself, we can in effect have a function return multiple things.
- To make this work, typically you declare the function's parameters as pointers, and pass addresses of variables rather than variables.
- The "sort of" of the title means that this isn't true pass by reference, as it exists in some other languages such as C++, but it can be used to more or less get the same effect.

  (Example.)

Slide 7

I/O in C — Preview

- You already know about a function to write output to "standard output", `printf`. Many options, allowing a lot of control over what's printed.
- How about input? Counterpart of `printf` is `scanf` (skim man page). Simple to use, though error detection is somewhat crude, and reading text strings can be hazardous.

Slide 8

Minute Essay

- TBA